# MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning

**Yao Lai** **Yao Mu** **Ping Luo** *
Department of Computer Science
The University of Hong Kong
{laiyao,muyao,pluo}@cs.hku.hk

## Abstract

Placement is an essential task in modern chip design, aiming at placing millions of circuit modules on a 2D chip canvas. Unlike the human-centric solution, which requires months of intense effort by hardware engineers to produce a layout to minimize delay and energy consumption, deep reinforcement learning has become an emerging autonomous tool. However, the learning-centric method is still in its early stage, impeded by a massive design space of size ten to the order of a few thousand. This work presents MaskPlace to automatically generate a valid chip layout design within a few hours, whose performance can be superior or comparable to recent advanced approaches. It has several appealing benefits that prior arts do not have. Firstly, MaskPlace recasts placement as a problem of learning pixel-level visual representation to comprehensively describe millions of modules on a chip, enabling placement in a high-resolution canvas and a large action space. It outperforms recent methods that represent a chip as a hypergraph. Secondly, it enables training the policy network by an intuitive reward function with dense reward, rather than a complicated reward function with sparse reward from previous methods. Thirdly, extensive experiments on many public benchmarks show that MaskPlace outperforms existing RL approaches in all key performance metrics, including wirelength, congestion, and density. For example, it achieves 60%-90% wirelength reduction and guarantees zero overlaps. We believe MaskPlace can improve AI-assisted chip layout design. The deliverables are released at laiyao1.github.io/maskplace.

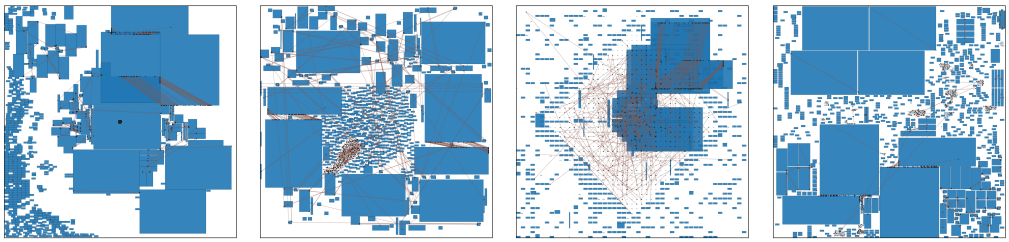
## 1 Introduction

The scalability and efficiency are two significant factors of autonomous chip layout design. Placement is one of the most challenging and time-consuming problems in the design flow, aiming to determine the locations of millions of circuit modules on a 2D chip canvas represented by a two-dimensional grid. A netlist can describe these modules, that is, a large-scale hypergraph consisting of massive macros (functional blocks such as memory) and standard cells (logic gates), where each macro and each standard cell can contain several or even hundreds of pins connected by wires, as shown in Fig.1.

Placing a large number of circuit modules onto the chip canvas is challenging because many performance metrics such as power consumption, timing, area, and wirelength should be minimized while satisfying some hard constraints such as placement density and routing congestion. For example, the wirelength (the length of wires that connect all modules) determines the delay and the power consumption of a chip [1]. Shorter wires often indicate less delay and less power consumption [2]. However, wirelength cannot be reduced by overlapping modules because the module density is a hard constraint to ensure that a valid and manufacturable chip layout has non-overlapping modules. More

*Corresponding author is Ping Luo

36th Conference on Neural Information Processing Systems (NeurIPS 2022).



(a) DREAMPlace [9] (b) Graph Placement [3] (c) DeepPR [22] (d) MaskPlace (ours)

Figure 1: **Visualizing different placements of a circuit benchmark** *bigblue3*, where the modules are visualized by blue rectangles and the wires are shown in brown lines to connect massive pins on modules. For clarity, we only show 1% wires. The proposed MaskPlace is compared with three representative approaches, including **(a)** DREAMPlace [9] (HPWL $= 1.04 \times 10^7$, WL $= 1.08 \times 10^7$, OL $= 8.06\%$), **(b)** Graph Placement [3] (HPWL $= 3.45 \times 10^7$, WL $= 3.73 \times 10^7$, OL $= 0.80\%$), **(c)** DeepPR [3] (HPWL $= 4.39 \times 10^7$, WL $= 5.18 \times 10^7$, OL $= 85.23\%$), and **(d) MaskPlace** (HPWL $= \underline{0.83 \times 10^7}$, WL $= \underline{0.88 \times 10^7}$, OL $= \underline{0\%}$), where HPWL, WL, and OL represent half-perimeter wirelength[2], wirelength, and overlap area ratio, respectively. All the metric values are smaller the better. The best performances are underlined in (d). We see that MaskPlace surpasses the recent popular placement approaches in all key metrics, and it can satisfy the $0\%$ hard density constraint. **Better zoom in 400%.**

examples of the performance metrics are given in Fig.8 and Fig.9 in Appendix. As pointed out in [3], the design space of placement is larger than $10^{2,500}$ when there are just $1,000$ circuit modules, whereas neural architecture search (NAS) typically has a space of $10^{30}$ and the Go game has a state space of $10^{360}$.

Methods of chip placement can be generally divided into two categories, classic optimization-based approaches [4–21] and learning-based approaches [3, 22, 23]. In the first category, hardware scientists often formulate placement as an optimization problem and relax the hard constraints. For example, let a pair of vectors $(\boldsymbol{x}, \boldsymbol{y})$ denote the $(x, y)$-coordinate value of all circuit modules on a 2D canvas, the objective function of placement can be formulated as minimizing $\mathrm{WL}(\boldsymbol{x}, \boldsymbol{y})$, subject to $\mathrm{D}(\boldsymbol{x}, \boldsymbol{y}) \leq \alpha$, where $\mathrm{WL}(\cdot, \cdot)$ and $\mathrm{D}(\cdot, \cdot)$ are the estimation functions of wirelength and density respectively, and $\mathrm{D}(\boldsymbol{x}, \boldsymbol{y}) \leq \alpha$ is a hard constraint with a very small density value $\alpha$, which ensures that all modules do not overlap. For instance, DREAMPlace [9] is a recent advanced method that minimizes $\mathrm{WL}(\boldsymbol{x}, \boldsymbol{y}) + \lambda \mathrm{D}(\boldsymbol{x}, \boldsymbol{y})$, which relaxes the hard density constraint. However, it cannot directly produce a valid and manufacturable layout because the non-overlapping constraint is not satisfied after relaxation. These approaches often need a post-processing step, such as manual refinement and legalization (LG), to remove the overlapping in placement, resulting in two issues, (1) the wirelength may increase substantially after LG, and (2) no feasible solution can be found if the available chip area is insufficient before post-processing.

In the second category, reinforcement learning (RL) is employed to solve placement as a sequential decision-making problem, placing each circuit module at a time. Although the learning-based approaches are still in their early stage, they can produce promising results to automate the chip design flow end-to-end significantly without human effort. For instance, Graph Placement [3] and DeepPR [22] represent a netlist as a hypergraph, denoted as $G = (V, E)$, where $V$ represents a set of nodes, and each node is a module, and $E$ is a set of edges, which are the wires connecting all modules. They train RL agents to place one module at a time by maximizing the metric values as rewards. However, the hypergraph is not scalable to comprehensively encode information of a netlist. For example, the relative positions (offsets) of pins are discarded in [3, 22]. The wirelength estimation is inaccurate without the pin information, but encoding this rich information would make the hypergraph too complicated because each module can have hundreds of pins. Furthermore, placement on a large hypergraph requires heavy computations. Mirhoseini et al. [3] reduced computations by placing 15% of the modules using reinforcement learning (the remaining modules are placed by classic method), and Cheng and Yan [22] decreased the size (resolution) of module and chip canvas as shown in Table 1. Both of them sacrificed their placement performance.

[2]HPWL (Half Perimeter Wire Length) is a common approximation metric of the wirelength and can be computed much more efficiently than wirelength.

Table 1: **Comparisons** of representative placement methods in different aspects, including method types ("Family"), canvas size ("Resolution"), state space, "0% overlap" (if the method can produce a layout without overlapping placement), training/inference speed ("Efficiency"), and the performance metrics to be optimized. We see that MaskPlace can outperform recent advanced methods by performing placement on a full canvas size of 224×224 (much larger than prior works) and producing a valid placement with 0% overlap (which cannot be achieved by previous methods). MaskPlace can also be trained and tested efficiently.

| | Family | Resolution | State Space | 0% Overlap | Reward | Efficiency | Metrics |
|---|---|---|---|---|---|---|---|
| DREAMPlace [9] | Nonlinear | Continuous | - | ✗ [1] | - | - /High | H, D [2] |
| Graph Placement [3] | RL+Nonlinear | $128^2$ | $(128^2)^{\alpha V}$ [3] | ✗ | Sparse | Med./Med. | H, C, D |
| DeepPR [22] | RL | $32^2$ | $(32^2)^V$ | ✗ | Dense | High/Med. | H, C |
| MaskPlace (ours) | RL | $224^2$ | $(224^2)^V$ | ✔ | Dense | High/High | H, C, D |

[1] DreamPlace needs a post-processing step, such as legalization (LG) that may fail.
[2] H = HPWL, C = Congestion, D = Density.
[3] $V$ is the number of circuit modules and $\alpha \approx 15\%$ in Graph Placement.

To address the issues of prior arts, we propose a novel RL method, named MaskPlace, which can automatically generate a high-quality and valid layout (non-overlapping modules) within a few hours, unlike previous methods that need manual refinement to modify invalid placement, which may wait up to 72 hours for commercial electronic design automation (EDA) tools to evaluate the placement. MaskPlace casts placement as a problem of pixel-level visual representation learning for circuit modules using convolutional neural networks. This representation can comprehensively capture the configurations of thousands of pins, enabling fast placement in a full action space on a large canvas size *e.g.,* 224×224. As shown in Fig.1 and Table 1, MaskPlace has many attractive benefits that existing works do not have. MaskPlace is mainly for macro placement due to the problem size.

This paper has three main **contributions**. Firstly, we recast chip placement as a problem of learning visual representation to describe millions of circuit modules on a chip comprehensively. It opens up a new perspective for AI-assist chip placement. Secondly, we carefully design a new policy network that can capture and aggregate both the global and subtle information on a chip canvas, maximizing the reward of wirelength and ensuring non-overlapping placement efficiently. Thirdly, extensive experiments demonstrate that MaskPlace outperforms recent advanced methods on 24 public chip benchmarks. For example, MaskPlace can always produce a layout with 0% overlap while reducing wirelength up to 5× and 9× compared to Graph Placement [3] and DeepPR [22] respectively.

## 2 Preliminary and Notation

The placement quality can be measured by the HPWL (half perimeter wirelength), which estimates the wirelength with marginal computational cost [24]. Intuitively, Fig.2(e) illustrates a 2D chip canvas. Let $M^i$ and $P^{(i,j)}$ denote the $i$-th module and its $j$-th pin, respectively. A net contains a set of pins connecting modules by wires. For example, "Net 1" (in red) connects all four modules (*i.e.,* $M^1, M^2, M^3, M^4$) using wires through pins $P^{(1,2)}$, $P^{(2,2)}$, $P^{(3,2)}$, and $P^{(4,1)}$, while "Net 2" (in green) connects three modules (*i.e.,* $M^1, M^2, M^3$) using wires through pins $P^{(1,1)}$, $P^{(2,1)}$, and $P^{(3,1)}$. HPWL estimates the wirelength by summing up the half perimeters of bounding boxes of all the nets, as shown by the red and green boxes in Fig.2(e). Intuitively, the half perimeter of a net bounding box equals the sum of its height and width. For example, HPWL in Fig.2(e) is $h_1 + w_1 + h_2 + w_2$.

Given a netlist containing a set of nets, minimizing the wirelength can be treated as minimizing HPWL by placing modules to the optimal positions on a 2D chip canvas. To achieve a valid and manufacturable chip layout, we need to satisfy two hard constraints: (1) *congestion constraint*: the wire congestion should be lower than a desired small threshold to reduce chip cost, and (2) *overlap constraint*: the density should be minimized to achieve non-overlapping placement.

$$\begin{aligned} \min \quad & \sum_{\forall \text{net} \in \text{netlist}} \Big( \max_{P^{(i,j)} \in \text{net}} P_x^{(i,j)} - \min_{P^{(i,j)} \in \text{net}} P_x^{(i,j)} + \max_{P^{(i,j)} \in \text{net}} P_y^{(i,j)} - \min_{P^{(i,j)} \in \text{net}} P_y^{(i,j)} \Big) \\ \text{s.t.} \quad & \text{Congestion}(M_x, M_y, M_w, M_h) \leq C_{\text{th}} \quad \text{and} \quad \text{Overlap}(M_x, M_y, M_w, M_h) = 0, \end{aligned} \tag{1}$$

where $P_x$ and $P_y$ represent the $(x, y)$-coordinate value of a pin respectively, $\text{Congestion}(\cdot)$ is the congestion function, $C_{\text{th}}$ is a desired threshold, $\text{Overlap}(\cdot)$ is the overlap function, and $M_x, M_y, M_w, M_h$

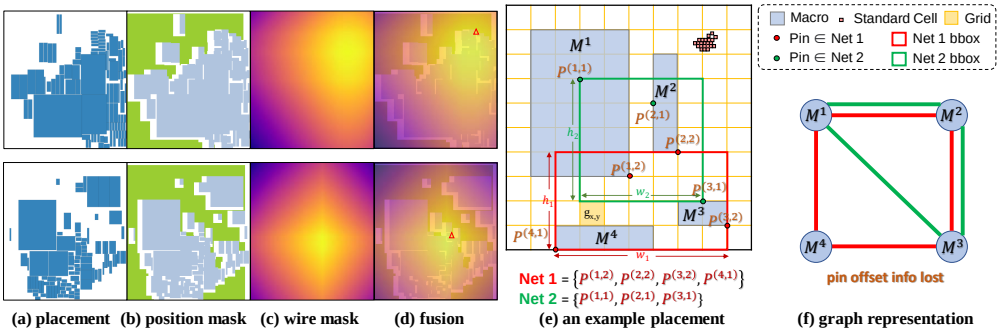



Figure 2: **Mask Visualization, placement example, and hypergraph representation in prior work.** We visualize different masks in MaskPlace (a-d) and illustrate an example of placement in (e). In the position mask (b), the green color means feasible positions to place while the gray color represents the placed modules. In the wire mask (c), lighter color indicates shorter wirelength if a module is placed at a specific position. The fusion mask in (d) is an example of the output after the mask fusion model using $1 \times 1$ convolutions, where the △ denotes the position with a high probability to place at (*i.e.,* no overlap and shorter wirelength). (f) is the result when converting the circuit in (e) into a hypergraph in prior works, where the critical information of pin locations is lost.

represent the position, width, and height of modules respectively. Firstly, lower congestion often indicates shorter wirelength, which is crucial to reduce chip cost because the wire resources are limited on a real chip. Inspired by prior arts [3, 22], we employ the RUDY estimator [25] to estimate wire congestion. Details of RUDY can be found in the Appendix A.2. Secondly, the placement density calculates the overlapping region between every pair of circuit modules. It is time-consuming since its computational complexity is $\mathcal{O}(V^2)$ where $V$ is the number of modules [1]. The proposed approach can ensure non-overlapping placement to avoid calculating this density metric explicitly in training, thus reducing computations while producing a valid layout.

## 3 Our Approach

**Model Architecture Overview.** Chip placement can be formulated as a Markov Decision Process (MDP) [26] by placing each module at a time. Fig.4 illustrates the overall architecture of MaskPlace, which trains a policy $\pi_\theta(a_t|s_t)$ represented by a convolutional encoder-decoder network with parameter set $\theta$, and a value function $V_\phi(s_t)$ represented by an embedding model with parameter set $\phi$. The policy network receives previous observations and actions as input $s_t$ and selects an action $a_t$ as output. Specifically, $s_t$ is a set of pixel-level feature maps that comprehensively capture the net and pin configurations in $M^{1:t-1}$, $M^t$, and $M^{t+1}$, where $M^{1:t-1}$ denotes the modules that have been placed in the previous time steps from 1 to $t-1$, while $M^t$ and $M^{t+1}$ denote the modules to be placed at the current step $t$ and the next step $t+1$, respectively. Intuitively, MaskPlace looks one step forward to achieve better placement.

Although prior arts [3, 22] represented a netlist as a hypergraph as shown in Fig.2(f) where each node is a module, and each edge is a wire between two modules, they lost the information of pin offsets for each module. Unlike previous works, MaskPlace can fully represent massive net and pin configurations using three types of pixel-level feature maps, as shown in Fig.2(a-d), including position mask, wire mask, and view mask, as discussed below. Different masks are fused by convolutions to learn the state representation.

**Position Mask.** The position mask, denoted by $f_p \in \{0,1\}^{224\times224}$, is a binary matrix of a canvas grid with size $224 \times 224$ as shown in Fig.3, where value "1" means a feasible position to place a module. The purpose of the position mask is to guarantee no overlaps between modules (*i.e.,* satisfy the overlap constraint) and to learn the relationship between placement and wirelength. Specifically, we slide a module $M^t$ (for example,

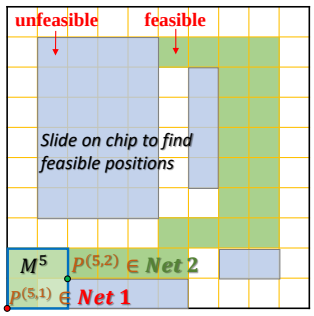



Figure 3: Position Mask Example.

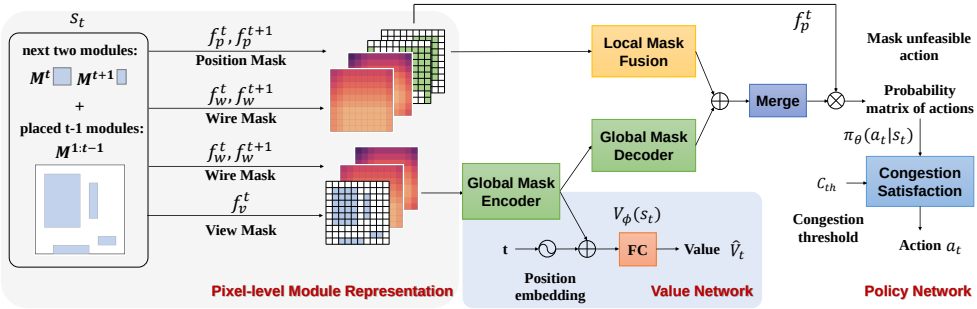



Figure 4: **Overview of MaskPlace,** which contains three main parts: a pixel mask generation model, a policy network, and a value network. The pixel mask generation model converts the current placement state into pixel-level masks. The policy and value networks convert these masks to actions and values based on global and local features. The congestion satisfaction block is to satisfy the congestion constraint and give the final action.

$t = 5$) on the entire chip canvas. The trajectory of the feasible positions (in green) can be labeled with "1". Intuitively, we can check each position for each module using the cumulative sum array [27]. This naive approach has the computational complexity of $\mathcal{O}(N^2)$ when a 2D canvas grid is divided into $N \times N$ cells. However, this simple approach is not efficient when $N$ is large. Therefore, since all modules are rectangles, we design an efficient generation algorithm, which iterates through all placed modules (in blue) and excludes positions that will cause overlap. In this case, all remaining positions are available for placement. The new algorithm is summarized in Appendix A.3, which costs $\mathcal{O}(V)$ for each module, where $V$ is the number of modules.

**Wire Mask.** The wire mask, denoted as $f_w \in [0,1]^{224\times224}$, is a continuous matrix for representing how HPWL increases if we place a module $M^t$ in a specific position. Fig.5 shows a sample of wire mask, where each value means the increase of HPWL. The wire mask aims at finding the best position with the minimum increase of the wirelength. Intuitively, we can calculate the HPWL at each canvas position, leading to a complexity of $\mathcal{O}(N^2P)$, where $P$ is the total number of pins. However, a fast algorithm can be designed by considering the relationships between the pin offset, the net bounding box, and the linear property of the HPWL metric. For example, Fig.3 illustrates that the next module $M^5$ has two pins, $P^{(5,1)}$ and $P^{(5,2)}$, belonging to "Net 1" and "Net 2" respectively (Fig.2(e)). Fig.5 illustrates the increase of wirelength when placing $M^5$ at each canvas location. For instance, if $M^5$ is at the bottom-left corner, its Manhattan distance to the two net bounding boxes (in red and green) is $2 + 2 = 4$. To calculate the Manhattan distance more accurately, we move the net bounding box compared to the pin location. For example, since $P^{(5,2)}$ is located at $(2, 1)$[3], we move the bounding box of Net 2 (in green) in the direction $-\Delta^{(5,2)} = (-2, -1)$ to encode the information of pin offset. The time complexity can be reduced to $\mathcal{O}(NP)$. The algorithm can be found in Appendix A.3.

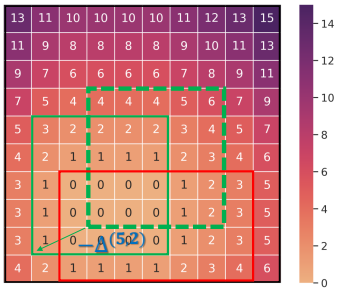



Figure 5: Wire Mask Example.

**View Mask.** The view mask, denoted as $f_v \in \{0,1\}^{224\times224}$, is a global observation of the current chip layout, where the value "1" means a module has occupied this grid cell. Different from DeepPR [22] that assumed all modules have unit size, we consider real sizes of modules. For instance, if a module has size $w \times h$, it covers $\lceil wN/W \rceil \times \lceil hN/H \rceil$ cells in the canvas, where $W$ and $H$ represent the canvas size and $\lceil \cdot \rceil$ denotes the ceiling function.

**Learning Algorithm.** We train different blocks in Fig.4 as a whole using reinforcement learning. The detailed network architectures are provided in Appendix A.4. Firstly, we apply the above masks to represent the entire circuits and feed them to downstream networks. Secondly, a global feature encoder embeds the view mask of current placement and the wire masks of the following two steps into an embedding vector. Then we combine it with the positional embedding of the $t$-th circuit module in the value network to generate a scalar to evaluate the current state by fully-connected layers.

[3]We index the bottom-left corner as the origin $(0, 0)$ in a two-dimensional coordinate.

Thirdly, a global mask decoder recovers a feature map of size $N^2$, which is fused with different position masks and wire masks in the policy network using $1 \times 1$ convolutions to avoid the local signal diffusion. The policy network predicts a probability action matrix of size $N \times N$, indicating where to put the next module. Before sampling actions, we can remove unfeasible actions using the position mask. Finally, the congestion satisfaction block applies the congestion threshold on the probability matrix to select a final action.

**Reinforcement Learning.** We borrow the representative actor-critic diagram [28] and PPO2 framework [29] to train the policy $\pi_\theta(a_t|s_t)$, where the state representation $s_t$ is listed in Table 11 in Appendix. The action $a_t$ is the canvas position (cell) to place the circuit module. Specifically, we treat the chip canvas as a grid and divide it into $N \times N$ cells, leading to $N^2$ possible actions. The objective function of the policy network can be formulated as

$$L_{\text{policy}}(\theta) = \hat{\mathbb{E}}\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(\text{r}_\text{t}(\theta), 1-\epsilon, 1+\epsilon)\hat{\text{A}}_\text{t}\big)\Big], \tag{2}$$

where the ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and $\hat{A}_t = G_t - \hat{V}_t$ denotes the advantage function. We employ $G_t = \sum_{k=0}^{V-t-1} \gamma^k r_{t+k+1}$ that is the cumulative discounted reward and $\hat{V}_t$ is the estimated value produced by the value network. We update the the value network by optimizing its objective, $L_{\text{value}}(\phi) = \hat{\mathbb{E}}\big[(G_t - \hat{V}_t)^2\big]$.

**Reward $r_t$.** We treat HPWL as the reward because wirelength is the main optimization target in different performance metrics. This is different from prior arts [3, 22] that weighted combines HPWL and congestion as the reward, which introduces the weighting coefficient as an extra hyper-parameter to tune. Specifically, we achieve a dense reward by defining a partial HPWL, which only computes the currently placed pins. For example, the partial HPWL for $t$ modules can be defined as $\text{HPWL}_t$. In other words, we compute $\text{HPWL}_t$ after taking action $a_t$. The reward for the step $t$ is $r_t = \text{HPWL}_{t-1} - \text{HPWL}_t$, *i.e.,* the opposite number of the increase of HPWL. Furthermore, instead of computing HPWL at each step, we can maintain the ranges of all net bounding boxes in one episode and update the changes of their sizes with a cost of $\mathcal{O}(P)$, where $P$ is the number of pins. Thus we can generate dense rewards while maintaining efficiency.

**Training and Testing.** Before training, we follow previous work [3] to sort the circuit modules according to the number of nets, areas, and the number of connected modules that have been placed to determine the place order. In training, we update the policy and value networks at each epoch by ignoring the congestion satisfaction block. When updating the value network, we stop the gradient back-propagated in the global mask encoder to avoid influence on the policy network. The detailed training setup is provided in Appendix A.5.

In the testing stage, for each step $t$, we obtain a probability matrix from the policy network and sample one place action $a_t$. Then, the congestion satisfaction block will check whether the congestion exceeds a threshold $C_{\text{th}}$ after applying this action. If it exceeds, we uniformly sample a few actions, look up the corresponding values from the wire mask $f_w^t$, and estimate the congestion before taking these actions. We choose the action with the minimal value in $f_w^t$ and the congestion lower than $C_{\text{th}}$. If all actions cannot satisfy congestion $C_{\text{th}}$, we select the action with the minimal congestion and move to the next step. Detailed congestion satisfaction algorithm can be seen in Appendix A.3.

## 4 Experiments

We extensively evaluate MaskPlace and compare it with several recent advanced placement methods, including NTUPlace3 [6], RePlAce [8], DREAMPlace [9], Graph Placement [3], and DeepPR [22], where NTUPlace3, RePlAce and DREAMPlace are optimization-based methods, whilst Graph Placement and DeepPR are learning-based approaches. All of them are evaluated on different public circuit benchmarks. All previous works are evaluated by following their experimental settings.

**Benchmark.** We evaluate MaskPlace in 24 circuit benchmarks selected from public datasets including the widely-used ISPD2005 [30], IBM benchmark suite [31], and Ariane RISC-V CPU design [32]. The number of evaluated benchmarks is three times more than previous work [9, 22, 3]. The statistics of benchmarks are given in Table 14 in Appendix A.6, where the largest circuit contains 1,293 macros, 22,802 pins, and more than a million standard cells, leading to a vast state space as aforementioned.

**Main Results.** Table 2 compares the HPWL results between all the above methods to place all macros. To enable a fair comparison, we evaluate all approaches[4] by using five random seeds and report the means and variances. Since the original DeepPR method did not capture macro size (thus does not avoid overlap between adjacent macros because all macros have unit size), we extend DeepPR to model macro size to reduce the overlapping ratio. We name it "DeepPR-no-overlap". Similar to prior works, we use the minimum spanning tree algorithm [33] to estimate routing wirelength [34]. From Table 2, we can see that MaskPlace achieves the lowest wirelength in six out of seven benchmarks (except "adaptec4" where it still outperforms all learning-centric methods). We also see that the conventional optimization-based approaches may fail when the circuit benchmark has high chip area usage, such as "bigblue3 " and "ariane". Also, MaskPlace gets the lowest wirelength compared with Graph Placement and simulated annealing [35] in the IBM benchmark, which is shown in Appendix A.7. This project website[5] visualizes and compares different placements.

Table 2: Comparisons of HPWL ($\times 10^5$). HPWL is the smaller the better. We see that MaskPlace outperforms other methods by large margins in six out of seven benchmarks. The traditional optimization such as NTUPlace3 and DREAMPlace may fail in a few benchmarks such as "ariane".

| Method | adaptec1 | adaptec2 | adaptec3 | adaptec4 | bigblue1 | bigblue3 | ariane |
|---|---|---|---|---|---|---|---|
| Random | 61.00±3.85 | 483.12±13.65 | 576.25±16.03 | 600.07±14.17 | 36.67±3.18 | 918.05±43.49 | 52.20±0.90 |
| NTUPlace3 [6] | 26.62 | 321.17 | 328.44 | 462.93 | 22.85 | 455.53 | LG fail |
| RePlAce [8] | 16.19±2.10 | 153.26±29.01 | 111.21±11.69 | 37.64±1.05 | 2.45±0.06 | 119.84±34.43 | LG fail |
| DREAMPlace [9] | 15.81±1.64 | 140.79±26.73[1] | 121.94±25.05 | **37.41±0.87** | 2.44±0.06 | 107.19±29.91[2] | LG fail |
| Graph Placement [3] | 30.10±2.98 | 351.71±38.20 | 358.18±13.95 | 151.42±9.72 | 10.58±1.29 | 357.48±47.83 | 16.89±0.60 |
| DeepPR [22] | 19.91±2.13 | 203.51±6.27 | 347.16±4.32 | 311.86±56.74 | 23.33±3.65 | 430.48±12.18 | 52.20±0.89 |
| DeepPR-no-overlap [22] | 47.39±4.02 | 425.86±19.59 | 545.40±16.40 | 525.51±10.85 | 26.29±1.48 | 815.10±40.36 | 62.82±0.82 |
| MaskPlace | **6.38±0.35** | **73.75±6.35** | **84.44±3.60** | 79.21±0.65 | **2.39±0.05** | **91.11±7.83** | **14.63±0.20** |

[1] 2 (of 5) seeds fail in legalization (LG).
[2] 1 (of 5) seed fails in legalization (LG).

**Compare to Graph Representation.** Since Graph Placement [3] is the recent advanced learning-based approach that employs hypergraph for placement, we compare MaskPlace with it in all four performance metrics, including HPWL, congestion, density, and overlap area ratio. Table 3 and 4 report the results. The overlap area ratio describes the ratio of the overlapping area between macros divided by the chip area. In Table 3, MaskPlace (soft constraint) means that the round function rather than the ceiling function is used to calculate the number of grid cells occupied by the placed macros. MaskPlace with soft constraints may produce better HPWL and congestion than its counterpart with hard constraints, but the overlap area ratio could not be zero because the constraints have been relaxed. From Table 3 and 4, we see that MaskPlace outperforms graph placement by large margins, especially in the ISPD benchmark, where the former reduces HPWL compared to the latter one by up to 80% while ensuring zero overlaps in all benchmarks. More results in the IBM benchmark can be found in Appendix Table 15.

Table 3: Comparisons between GraphPlace [3] and the proposed MaskPlace using different performance metrics (normalized to $[0, 1]$) in the "ariane" benchmark, including HPWL, congestion, density, and overlap area ratio. All values are smaller the better. We see that MaskPlace surpasses other methods significantly while ensuring zero overlaps, which is essential for a valid and manufacturable chip layout.

| Method | HPWL | Congestion | Density | Overlap (%) |
|---|---|---|---|---|
| Graph Placement (journal) [3] | 0.1198±0.0019 | 0.9718±0.0346 | 0.5729±0.0086 | 5.13±0.11 |
| Graph Placement (github) [3] | 0.1013±0.0036 | 0.9174±0.0647 | 0.5502±0.0568 | 4.29±1.25 |
| MaskPlace (hard constraint) | 0.1025±0.0015 | 1.0137±0.0451 | **0.5000±0.0000** | **0.00±0.00** |
| MaskPlace (soft constraint) | **0.0879±0.0012** | **0.9049±0.0115** | 0.5262±0.0015 | 3.33±0.79 |

**Routing Wirelength.** Table 5 compares the routing wirelength between MaskPlace and DeepPR [22]. We show that using the true wirelength as the reward would lower the efficiency and produce a sparse reward. We see that MaskPlace, which employs HPWL as the reward, can achieve 60% to 90% shorter routing wirelength than DeepPR, which directly used the true wirelength as the reward.

**Standard Cells.** Table 6 compares the HPWL of both the macros and the standard cells by using MaskPlace, DeepPR [22], and DREAMPlace [9], where DREAMPlace is employed to place the standard cells following the experimental setup in [22]. We can see that the proposed method

[4] The random seed does not apply in a classic method NTUPlace3.
[5] laiyao1.github.io/maskplace

Table 4: Comparisons between GraphPlace [3] and MaskPlace in four performance metrics (normalized to $[0, 1]$) in the ISPD benchmark. All values are smaller the better. We see that MaskPlace can reduce the HPWL up to 80% compared to its counterpart while ensuring the modules' overlaps are zeros in all benchmarks.

| benchmark | Graph Placement [3] | | | | MaskPlace | | | |
|---|---|---|---|---|---|---|---|---|
| | HPWL | Congestion | Density | Overlap(%) | HPWL | Congestion | Density | Overlap (%) |
| adaptec1 | 0.1810 | 0.7370 | 0.5340 | 1.89 | **0.0384** | **0.6961** | **0.5000** | **0.00** |
| adaptec2 | 0.2814 | 0.7387 | 0.5147 | 1.54 | **0.0549** | **0.6990** | **0.5000** | **0.00** |
| adaptec3 | 0.2248 | 0.7431 | 0.5226 | 1.24 | **0.0540** | **0.7130** | **0.5000** | **0.00** |
| adaptec4 | 0.1107 | 0.7369 | 0.7472 | 7.59 | **0.0560** | **0.7078** | **0.5000** | **0.00** |
| bigblue1 | 0.0958 | 0.7346 | 0.5181 | 1.98 | **0.0255** | **0.6953** | **0.4876** | **0.00** |
| bigblue3 | 0.1565 | 0.7499 | 0.5174 | 0.96 | **0.0430** | **0.7350** | **0.5000** | **0.00** |

Table 5: Compare routing wirelength ($\times 10^5$) between DeepPR [22] and MaskPlace.

| method | adaptec1 | adaptec2 | adaptec3 | adaptec4 | bigblue1 | bigblue3 | ariane |
|---|---|---|---|---|---|---|---|
| DeepPR [22] | 23.25±3.03 | 212.97±5.84 | 377.80±5.49 | 367.57±64.44 | 28.51±3.90 | 507.39±14.82 | 56.77±0.87 |
| DeepPR-no-overlap [22] | 52.46±3.97 | 451.22±19.00 | 583.32±15.92 | 628.22±10.02 | 31.02±1.41 | 945.60±43.24 | 68.89±0.81 |
| MaskPlace | **7.12±0.34** | **77.70±6.77** | **90.40±3.82** | **92.51±0.38** | **2.81±0.51** | **103.24±10.48** | **15.61±0.19** |

surpasses the other approaches by up to 50% in the large benchmark "bigblue3", which has more than a million standard cells.

Table 6: Comparisons of HPWL ($\times 10^7$) for macro and standard cell placement.

| Method | adaptec1 | adaptec2 | adaptec3 | adaptec4 | bigblue1 | bigblue3 |
|---|---|---|---|---|---|---|
| DREAMPlace [9] | 11.01±1.37 | 16.19±2.60 | 21.54±1.19 | 35.47±4.97 | 10.28±1.11 | 70.02±46.06 |
| DeepPR [22] + DREAMPlace [9] | 8.01 | 12.32 | 24.11 | 23.64 | 14.04 | 45.06 |
| MaskPlace + DREAMPlace [9] | **7.93±0.20** | **9.95±0.29** | **21.49±0.90** | **22.97±0.92** | **9.43±0.13** | **37.29±0.67** |

**Placement w/o real size** Considering that DeepPR ignored the module size, we implement MaskPlace in the same setting, and the result can be found in Table 7. The result shows that our method has significant advantages.

Table 7: Routing wirelength for macro placement w/o real size

| Method | adaptec1 | adaptec2 | adaptec3 | adaptec4 | bigblue1 | bigblue3 |
|---|---|---|---|---|---|---|
| DeepPR [22] | 5298 | 22256 | 32839 | 63560 | 8602 | 94083 |
| MaskPlace | **2941** | **20593** | **16181** | **18553** | **2331** | **27403** |

**Transferability** Test the performance of the model trained on *adaptec1* on other benchmarks as Table 8. The results show that our method also has a good transferability.

Table 8: HPWL ($\times 10^5$) results for transferability. HPWL is the smaller the better. The model has been trained on *adaptec1* benchmark and just took the inference in other benchmarks.

| | adaptec2 | adaptec3 | adaptec4 | bigblue1 | bigblue3 | ariane |
|---|---|---|---|---|---|---|
| HPWL | 85.56±9.41 | 89.77±6.72 | 87.32±3.93 | 2.87±0.31 | 160.63±10.41 | 19.32±2.02 |
| ratio* | 1.16 | 1.06 | 1.11 | 1.20 | 1.76 | 1.32 |

* Compared with the result from the model trained on the corresponding benchmark.

**Efficiency.** Table 9 compares the inference efficiency of different approaches. All of them are evaluated on one GeForce RTX 3090 GPU, and the CPU version of DREAMPlace is allocated with 16 threads in a 16 CPU cores environment. We see that the careful design of MaskPlace makes it faster than two other learning-based approaches.

**Ablation Study.** We compare different components in MaskPlace as shown in Fig.6. Each curve is produced by five seeds using the benchmark "adaptec1". For example, MaskPlace *w/ CL* means using 1/3 of the circuit macros to pretrain the model for 30 epochs like curriculum learning. MaskPlace *w/o* $M^{t+1}$ means only considering $M^t$ as input without looking one step forward. Moreover, MaskPlace

Table 9: Comparisons of wall-clock runtime (second) of different placement methods in inference.

| Method | adaptec1 | adaptec2 | adaptec3 | adaptec4 | bigblue1 | bigblue3 |
|---|---|---|---|---|---|---|
| DREAMPlace (CPU) [9] | 4.47 | 11.50 | 11.52 | 15.55 | 9.32 | 27.36 |
| DREAMPlace (GPU) [9] | 4.51 | 7.57 | 7.70 | **7.39** | 5.57 | **12.25** |
| Graph Placement [3] | 6.32 | 16.97 | 20.05 | 13.40 | 4.54 | 15.65 |
| DeepPR [22] | 10.25 | 10.46 | 22.82 | 42.24 | 9.86 | 32.53 |
| MaskPlace | **4.26** | **6.98** | **7.63** | 13.36 | **4.32** | 13.87 |

*w/o number of nets* means this feature is not considered when determining the placement order. MaskPlace *w/o 1x1 conv* means that 7x7 kernels are used to replace the 1x1 kernels in the local feature fusion block. Also, MaskPlace with *sparse reward* means compute HPWL reward only when all macros have been placed, and the rewards for other steps are set to zeros. MaskPlace *w/o view mask* and *w/o wire mask* means that the corresponding mask is not inputted into the model. We can see that MaskPlace (standard) with curriculum learning has the best performance.

**Congestion Satisfaction.** To evaluate our congestion satisfaction block, we implement a placement without any congestion threshold (*i.e.,* $\infty$) as shown in Fig.7. We evaluate the "adaptec3" benchmark, where MaskPlace outperforms DeepPR. We gradually lower the threshold $C_{th}$ from 60 to 10. We find that lower congestion leads to an increase in the HPWL. Our method can always satisfy the congestion constraint in five seeds in a suitable range (above 40 in this benchmark). If we continue to reduce the congestion threshold after a specific value (say 40 in Fig.7), we found that it hardly satisfies the threshold because nets must take up a certain amount of wire resources.

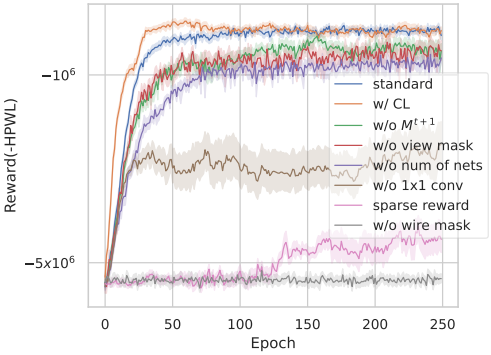



Figure 6: Compare reward curves of different components in MaskPlace.

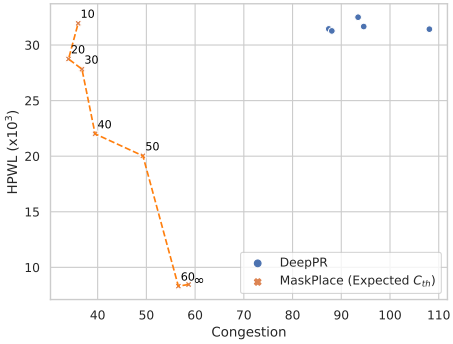



Figure 7: Study of congestion satisfaction.

## 5 Conclusion

This paper proposes MaskPlace, an RL-based placement method based on rich visual representation by learning position, wirelength, and view information. It helps the model take action effectively and efficiently without reducing the search space. We design a direct reward function based on practical scenarios and get satisfactory results on all key metrics. This work can facilitate the placement process and avoid undesired overlaps between modules. In the future, we will explore the standard cell placement by designing a suitable representation, which is an open problem for RL due to its vast space.

**Limitation and Potential Negative Societal Impact.** Chip design flow contains many stages, and our method shows its potential in a single stage. Similar to previous RL methods, it also requires an optimization method when placing millions of standard cells because RL's state space is too large. Our method does not have potential harm to the public society at the moment.

## Acknowledgments and Disclosure of Funding

We thank Xibo Sun from Huawei answering questions about EDA. Ping Luo is supported by the General Research Fund of HK No.27208720, No.17212120, and No.17200622.

## References


[1] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[2] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Digital integrated circuits*. Prentice hall Englewood Cliffs, 2002, vol. 2.

[3] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[4] J. A. Roy, S. N. Adya, D. A. Papa, and I. L. Markov, "Min-cut floorplacement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1313–1326, 2006.

[5] A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh, and P. H. Madden, "Recursive bisection based mixed block placement," in *Proceedings of the 2004 international symposium on Physical design*, 2004, pp. 84–89.

[6] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1228–1240, 2008.

[7] J. Lu, P. Chen, C.-C. Chang, L. Sha, J. Dennis, H. Huang, C.-C. Teng, and C.-K. Cheng, "eplace: Electrostatics based placement using nesterov's method," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[8] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1717–1730, 2018.

[9] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 748–761, 2020.

[10] X. Yang, M. Sarrafzadeh *et al.*, "Dragon2000: Standard-cell placement tool for large industry circuits," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD-2000. IEEE/ACM Digest of Technical Papers (Cat. No. 00CH37140)*. IEEE, 2000, pp. 260–263.

[11] D. Vashisht, H. Rampal, H. Liao, Y. Lu, D. Shanbhag, E. Fallon, and L. B. Kara, "Placement in integrated circuits using cyclic reinforcement learning and simulated annealing," *arXiv preprint arXiv:2011.07577*, 2020.

[12] N. Viswanathan, G.-J. Nam, C. J. Alpert, P. Villarrubia, H. Ren, and C. Chu, "Rql: Global placement via relaxed quadratic spreading and linearization," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 453–458.

[13] N. Viswanathan, M. Pan, and C. Chu, "Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *2007 Asia and South Pacific Design Automation Conference*. IEEE, 2007, pp. 135–140.

[14] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov, and S. Ramji, "Maple: Multilevel adaptive placement for mixed-size designs," in *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, 2012, pp. 193–200.

[15] M.-C. Kim and I. L. Markov, "Complx: A competitive primal-dual lagrange optimization for global placement," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 747–752.

[16] U. Brenner, A. Hermann, N. Hoppmann, and P. Ochsendorf, "Bonnplace: A self-stabilizing placement framework," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, 2015, pp. 9–16.

[17] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev, "Polar: Placement based on novel rough legalization and refinement," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013, pp. 357–362.

[18] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1398–1411, 2008.

[19] T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze, and M. Xie, "mpl6: Enhanced multilevel mixed-size placement," in *Proceedings of the 2006 international symposium on Physical design*, 2006, pp. 212–214.

[20] A. B. Kahng, S. Reda, and Q. Wang, "Aplace: A general analytic placement framework," in *Proceedings of the 2005 international symposium on Physical design*, 2005, pp. 233–235.

[21] J. Gu, Z. Jiang, Y. Lin, and D. Z. Pan, "Dreamplace 3.0: Multi-electrostatics based robust vlsi placement with region constraints," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[22] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[23] Z. Jiang, E. Songhori, S. Wang, A. Goldie, A. Mirhoseini, J. Jiang, Y.-J. Lee, and D. Z. Pan, "Delving into macro placement with reinforcement learning," in *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2021, pp. 1–3.

[24] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "A high-quality mixed-size analytical placer considering preplaced blocks and density constraints," in *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, 2006, pp. 187–192.

[25] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[27] Z. Guo, J. Mai, and Y. Lin, "Ultrafast cpu/gpu kernels for density accumulation in placement," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1123–1128.

[28] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[30] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, "The ispd2005 placement contest and benchmark suite," in *Proceedings of the 2005 international symposium on Physical design*, 2005, pp. 216–220.

[31] S. Adya, S. Chaturvedi, and I. Markov, "Iccad'04 mixed-size placement benchmarks," *GSRC Bookshelf*, 2009.

[32] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[34] H. Liao, Q. Dong, X. Dong, W. Zhang, W. Zhang, W. Qi, E. Fallon, and L. B. Kara, "Attention routing: track-assignment detailed routing using attention-based reinforcement learning," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 84003. American Society of Mechanical Engineers, 2020, p. V11AT11A002.

[35] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[36] J. Yan, X. Lyu, R. Cheng, and Y. Lin, "Towards machine learning for placement and routing in chip design: a methodological overview," *arXiv preprint arXiv:2202.13564*, 2022.

[37] Y.-H. Huang, Z. Xie, G.-Q. Fang, T.-C. Yu, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routability-driven macro placement with embedded cnn-based prediction model," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 180–185.

[38] R. Kirby, K. Nottingham, R. Roy, S. Godil, and B. Catanzaro, "Guiding global placement with reinforcement learning," *arXiv preprint arXiv:2109.02631*, 2021.

[39] A. Agnesina, K. Chang, and S. K. Lim, "Vlsi placement parameter optimization using deep reinforcement learning," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[40] F.-C. Chang, Y.-W. Tseng, Y.-W. Yu, S.-R. Lee, A. Cioba, I.-L. Tseng, D.-s. Shiu, J.-W. Hsu, C.-Y. Wang, C.-Y. Yang *et al.*, "Flexible multiple-objective reinforcement learning for chip placement," *arXiv preprint arXiv:2204.06407*, 2022.

[41] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

## Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

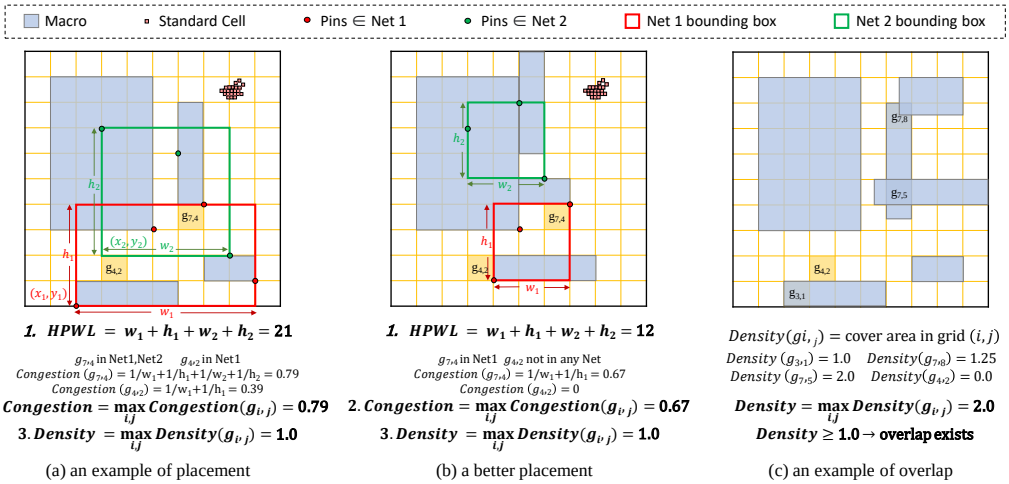
# A Appendix

## A.1 Module, Net and Pin

**Module.** A chip is a combination of numerous modules, and there are two types of them: macros and standard cells. Macros are relatively large, including DRAMs, caches, and IO interfaces. Standard cells are mainly logical gates, much smaller than macros, and the size can be ignored. As in Fig.8 (a), there are four macros and several standard cells. Placement methods usually place macros first and then the standard cells to ensure there is enough space for macros [36]. Due to the considerable number of standard cells, we currently use our MaskPlace method on macro placement.

**Pin.** Pins are input/output interfaces for modules and are connected by wires directly, which have fixed relative positions on modules. We define the relative position of the pin $P^{(i,j)}$ from the left-bottom corner of the module it belongs to as $\Delta^{(i,j)} = (\Delta_x^{(i,j)}, \Delta_y^{(i,j)})$. For example, there are five pins and three macros in Fig.9 (a), and the pin offset information is also shown at the bottom. In the placement task, we should not ignore the positions of pins because it determines the wirelength. However, graph neural network-based models [3, 22] ignored them when converting circuits into a graph, which may lead to sub-optimal results.

**Net.** A net contains a set of pins connected by the same wires. Thus the pins have the same information (0/1 in digital circuits). For example, four pins belong to Net 1, and the other three pins belong to Net 2 in Fig.8 (a). Usually, one pin belongs to only one net, and one net has more than two pins (one input and several outputs). Pins from the same net can form a net bounding box as Fig.8 (a)(b).




(a) an example of placement (b) a better placement (c) an example of overlap

Figure 8: **Metrics for placement.** HPWL is an optimization item, while congestion and density are constraint items in the actual placement scenario. HPWL is smaller the better, while congestion and density need to be less than the given thresholds. Placement (b) is better than (a) because HPWL and congestion of (a) are smaller. Placement (c) is invalid because there are overlaps in cell $g_{7,5}$ and $g_{7,8}$.

## A.2 Metric

**HPWL.** HPWL (Half Perimeter Wire Length) is widely used to estimate wirelength by small computation cost [24]. It is the sum of half perimeter of net bounding boxes as Fig.8 (a)(b), where the bounding box is the minimal rectangle including all pins belonging to this net.

**Congestion.** The congestion metric is used to avoid routing congestion, resulting in an increase in the actual wirelength because the resources for wires are limited in a real chip. There are many ways to estimate congestion, one is to compute a rough routing result [3], but it is very computationally intensive. We use RUDY [25] as the estimation of congestion, which is a common way to evaluate. In RUDY, each grid cell needs to accumulate the inverse of the height and width $(1/h + 1/w)$ of all

the net bounding boxes covering itself and take out the maximum value (or the average of the first k maximums) of all grid cells as Fig.8 (a)(b).

**Density.** Density is a metric to reduce overlaps and avoid time-consuming computation for $O(V^2)$ constraints [1]. So, it is an approximate calculation essentially. It is defined as the maximum stackable coverage area ratio for each grid cell on a chip canvas. For example, as Fig.8 (c), the maximum stackable coverage area ratio is 2.0 in grid cell $g_{7,5}$ because two modules fully occupy it. However, density less than a small value is not a sufficient condition for the absence of overlap. Because our method can ensure no overlaps, we only consider it in evaluation. In the practical application scenario of chip design, HPWL is an optimization item. Conversely, congestion and density are constraint items.

**Examples.** We give a set of placement results to explain the metrics in Fig.8. We can see that HPWL is the sum of width and height of net bounding boxes. Congestion (RUDY) is the max congestion value of grid cell $g_{i,j}$, and the value in each grid cell is cumulative from the reciprocal of the width and height of the net bounding box containing that grid cell. (a) and (b) are from the same circuit, but (b) is a better placement because (b) has lower HPWL and congestion. Density is the max density value of grid cell $g_{i,j}$, and the value in each grid cell is stackable coverage area ratio of the grid cell. The density of Fig.8 (c) is 2.0 because $g_{7,5}$ completely covered by two modules.

**Relationship between pin offset and HPWL.** The pin offset can affect the HPWL. In the graph-based method, the input features for module include size $(M_w, M_h)$, position $(M_x, M_y)$ and type. So, the network can hardly infer the real position of pins and tend to use the center positions of modules to predict the positions of pins. In this way, the agent will align the centers of the two modules horizontally, and the result of placement is like Fig.9 (b) to get the wirelength 6. However, when considering the pins are near the bottom of the modules, it is better to align the bottom of the two modules as Fig.9 (c), and thus wirelength can be reduced to 2 if we consider the pin offset.

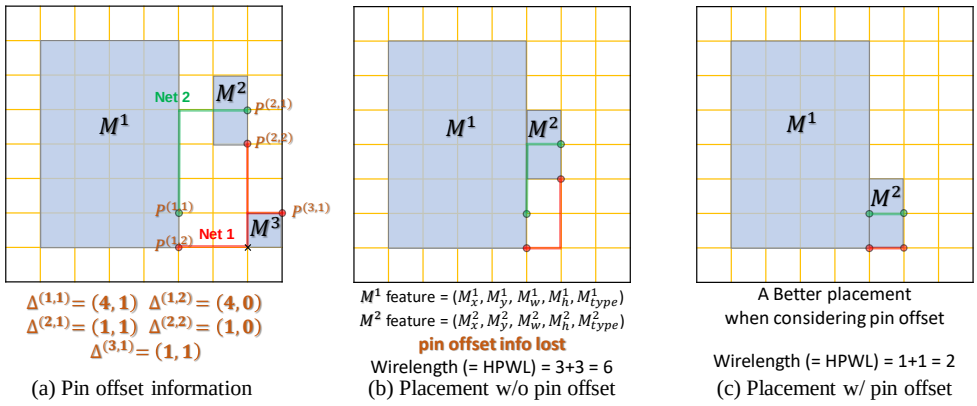



Figure 9: **Explanation for module, pin and net.** (a) gives an example for pin offset information. When we remove the pin offset information, the model tends to align the centers of the two modules horizontally like (b) because it uses the center position of modules to estimate pin location. However, we have a better design as (c) when considering pins are located on the bottom of the modules.

### A.3 Algorithms

**Reward Computation.** The dense reward generation algorithm is shown in Algorithm 1. It can generate dense rewards without an efficiency decrease. For simplicity, we omit the calculation of the y dimension, which is the same as the x dimension.

**Position Mask Generation.** The efficient position mask generation algorithm is in Algorithm 2.

**Wire Mask Generation.** The efficient wire mask generation algorithm is shown in Algorithm 3. For simplicity, we omit the calculation of the y dimension, which is the same as the x dimension.

**Algorithm 1:** Dense HPWL Reward Computation (omit y-dimension)

**Data:** Placed position of module $M^t$ $(M_x^t, M_y^t)$, max/min x/y coordinates of nets $MaxMinCoord$, pin offsets $(\Delta_x^{(t,j)}, \Delta_y^{(t,j)})$, pin to net connection $P_n^{(t,j)}$;
**Result:** Incremental HPWL Reward $reward$;
$reward \leftarrow 0$;
**foreach** $\Delta_x^{(t,j)}, P_n^{(t,j)}$ *of all pins* $P^{(t,j)}$ *from* $M^t$ **do**
  $x \leftarrow M_x^t + \Delta_x^{(t,j)}$ ; `// calculate pin coordinates`
  **if** $P_n^{(t,j)}$ *not in* $MaxMinCoord$ **then**
    `// The net for the first time has a definite location of the pin`
    $MaxMinCoord[P_n^{(t,j)}].x.max \leftarrow x$;
    $MaxMinCoord[P_n^{(t,j)}].x.min \leftarrow x$;
  **else**
    `// Update the bounding box range`
    **if** $MaxMinCoord[P_n^{(t,j)}].x.max < x$ **then**
      $reward \leftarrow reward + (x - MaxMinCoord[P_n^{(t,j)}].x.max)$;
      $MaxMinCoord[P_n^{(t,j)}].x.max = x$;
    **else if** $MaxMinCoord[P_n^{(t,j)}].x.min > x$ **then**
      $reward \leftarrow reward + (MaxMinCoord[P_n^{(t,j)}].x.min - x)$ ;
      $MaxMinCoord[P_n^{(t,j)}].x.min = x$;
    **end**
  **end**
**end**

**Algorithm 2:** Position Mask Generation

**Data:** Width, Height and Position of t-1 placed module $M^{1:t-1}$ $(M_w^{1:t-1}, M_h^{1:t-1}, M_x^{1:t-1}, M_y^{1:t-1})$
**Result:** Position Mask $f_p^t$ for Module $M^t$
$f_p^t \leftarrow ones(N, N)$; `//` $ones(N, N)$ `is all-ones` $N \times N$ `matrix`
**for** $i \leftarrow 1$ **to** $t-1$ **do**
  $tmp \leftarrow ones(N, N)$;
  `// find positions that will cause` $M^t$ `and` $M^i$ `to overlap`
  $tmp[M_x^i - M_w^t + 1 : M_x^i + M_w^i - 1, M_y^i - M_h^t + 1 : M_y^i + M_h^i - 1] \leftarrow 0$;
  `// exclude infeasible positions`
  $f_p^t \leftarrow tmp \odot f_p^t$ ; `//` $\odot$ `is element-wise product`
**end**

**Algorithm 3:** Wire Mask Generation (omit y-dimension)

**Data:** Hash Map of Max/Min X/Y coordinates of nets $MaxMinCoord$, pin's offsets $(\Delta_x^{(t,j)}, \Delta_y^{(t,j)})$, pin to net connection $P_n^{(t,j)}$
**Result:** Wire Mask $f_w^t$ for module $M^t$
$f_w^t \leftarrow zeros(N, N)$;
// Accumulate the wirelength increase for each net
**foreach** $\Delta_x^{(t,j)}, P_n^{(t,j)}$ *of all pins* $P^{(t,j)}$ *from* $M^t$ **do**
  // If the pin is to the left of the net bounding box
  **for** $i \leftarrow 0$ **to** $MaxMinCoord[P_n^{(t,j)}].x.min + \Delta_x^{(t,j)} - 1$ **do**
    $f_w^t[i,:] \leftarrow f_w^t[i,:] + MaxMinCoord[P_n^{(t,j)}].x.min + \Delta_x^{(t,j)} - i$;
  **end**
  // If the pin is to the right of the net bounding box
  **for** $i \leftarrow MaxMinCoord[P_n^{(t,j)}].x.max + \Delta_x^{(t,j)} + 1$ **to** $N - 1$ **do**
    $f_w^t[i,:] \leftarrow f_w^t[i,:] + i - (MaxMinCoord[P_n^{(t,j)}].x.max + \Delta_x^{(t,j)})$;
  **end**
**end**

**Congestion Satisfaction.** The algorithm implemented in the congestion satisfaction block can be seen in Algorithm 4.

**Algorithm 4:** Placement with Congestion Constraint

**Data:** Trained place agent $agent$, expected congestion threshold $C_{th}$
**Result:** A placement plan $[a_1, a_2, ..., a_V]$ that meet the congestion requirement
**for** $i \leftarrow 1$ **to** $V$ **do**
  Choose $a_i$ from the probability matrix generated by policy network $agent$;
  $Cong \leftarrow$ congestion matrix from the state after taking $a_i$;
  Compute congestion value $c$ from $Cong$;
  **if** $c > C_{th}$ **then**
    Randomly sample $N$ different actions $a_i^{1:N}$ from action space;
    Compute $N$ congestion values $c_i^{1:N}$ from congestion metrics;
    Get $N$ wirelength values $w_i^{1:N}$ from wire masks;
    Sort the $N$ actions according to $w_i^{1:N}$ (as the 1st key) and $c_i^{1:N}$ (as the 2nd key);
    $flag \leftarrow False$ ;
    **for** $j \leftarrow 1$ **to** $N$ **do**
      **if** $c_i^j \leq C_{th}$ **then**
        $flag \leftarrow True$;
        $a_i \leftarrow a_i^j$;
        break;
      **end**
    **end**
    // If all sampled actions cannot satisfy congestion threshold, we choose the one with minimal congestion increase.
    **if** $flag$ *is* $False$ **then** $a_i \leftarrow$ the action $a_i^j$ with minimum $c_i^j$;
  **end**
  Take action $a_i$ as the final action;
**end**

### A.4 Details of Model Architecture

The parameters of layers in model architecture are in Table 10. Also, the features used by pixel-level mask generation are in Table 11. The comparison of features for the placement order in different methods can be seen in Table 12.

Table 10: Model Architecture

| Block | Layer | Kernel Size | Output shape |
|---|---|---|---|
| Local Mask Fusion | Conv | $1 \times 1$ | $(224, 224, 8)$ |
| | Conv | $1 \times 1$ | $(224, 224, 8)$ |
| | Conv | $1 \times 1$ | $(224, 224, 1)$ |
| Global Mask Encoder | ResNet-18 | - | 1000 |
| | FC | - | 768 |
| Global Mask Decoder | Deconv | $3 \times 3$ | $(14, 14, 8)$ |
| | Deconv | $3 \times 3$ | $(28, 28, 4)$ |
| | Deconv | $3 \times 3$ | $(56, 56, 2)$ |
| | Deconv | $3 \times 3$ | $(112, 112, 1)$ |
| | Deconv | $3 \times 3$ | $(224, 224, 1)$ |
| Merge | Conv | $1 \times 1$ | $(224, 224, 1)$ |
| Position Embedding | - | - | 64 |
| FC for Value | FC | - | 512 |
| | FC | - | 64 |
| | FC | - | 1 |

Table 11: State Features

| Module status | Index | Feature | Notation | Dimension per module |
|---|---|---|---|---|
| Placed | $M^{1:t-1}$ | Width | $M_w$ | 1 |
| | | Height | $M_h$ | 1 |
| | | Position | $M_x, M_y$ | 2 |
| | | Pin Offset | $\Delta_x, \Delta_y$ | $2 \times$ num of pins |
| | | Pin to Net Connection | $P_n$ | num of pins |
| Unplaced | $M^t, M^{t+1}$ | Width | $M_w$ | 1 |
| | | Height | $M_h$ | 1 |
| | | Pin Offset | $\Delta_x, \Delta_y$ | $2 \times$ num of pins |
| | | Pin to Net Connection | $P_n$ | num of pins |

Table 12: Features used for placement order

| Method | Features for place order |
|---|---|
| Graph Placement [3] | Topological order, Area |
| DeepPR [22] | None |
| MaskPlace | Number of nets, Area, Number of its connected modules have been placed |

### A.5 Training Configuration

The detailed configuration and hyperparameter settings of our model is in Table 13.

Table 13: Model Configuration

| Configuration | Value | Configuration | Value |
|---|---|---|---|
| Optimizer | Adam | Learning rate | $2.5 \times 10^{-3}$ |
| Total epoch | 150 | Epoch for update | 10 |
| Batch size | 64 | Buffer capacity | $10 \times$ num of modules |
| Clip $\epsilon$ | 0.2 | Clip gradient norm | 0.5 |
| Reward discount $\gamma$ | 0.95 | Num GPUs | 1 |
| CPU | AMD Ryzen 9 5950X | GPU | GeForce RTX 3090 |

Also, we implement DREAMPlace[6] [9], Graph Placement[7] [3] ,and DeepPR[8] [22] by their open source codes with their default settings.

### A.6 Details of Benchmark

The detailed statistics of benchmarks are in Table 14. Hard macros are macros placed by the RL method in Graph Placement [3], and the remaining macros, also named soft macros, are placed by the classic optimization-based method. However, this distinction does not apply to the process of our method, which means we place all macros by RL. The statistical range of nets, pins, and area utilization are macros. Ports are terminals connecting to an external circuit, seen as fixed and no-size modules. Our method is also applicable to circuits with ports without additional modifications.

Table 14: Statistics of different chip benchmarks.

| Benchmark | Macros | Hard Macros | Standard Cells | Nets | Pins | Ports | Area Util(%) |
|---|---|---|---|---|---|---|---|
| adaptec1 | 543 | 63 | 210,904 | 3,709 | 4,768 | 0 | 55.62 |
| adaptec2 | 566 | 190 | 254,457 | 4,346 | 10,663 | 0 | 74.46 |
| adaptec3 | 723 | 201 | 450,927 | 6,252 | 11,521 | 0 | 61.51 |
| adaptec4 | 1,329 | 92 | 494,716 | 5,939 | 13,720 | 0 | 48.62 |
| bigblue1 | 560 | 32 | 277,604 | 657 | 1,897 | 0 | 31.58 |
| bigblue3 | 1,293 | 138 | 1,095,519 | 5,537 | 15,225 | 0 | 66.81 |
| ariane | 932 | 134 | 0 | 12,404 | 22,802 | 1,231 | 78.39 |
| ibm01 | 246 | 246 | 12,506 | 908 | 1,928 | 246 | 61.94 |
| ibm02 | 280 | 272 | 19,321 | 602 | 1,466 | 259 | 64.63 |
| ibm03 | 290 | 290 | 22,846 | 614 | 1,237 | 283 | 57.97 |
| ibm04 | 608 | 296 | 26,899 | 1,512 | 3,167 | 287 | 54.88 |
| ibm06 | 178 | 178 | 32,320 | 83 | 175 | 166 | 54.77 |
| ibm07 | 507 | 292 | 45,419 | 2,471 | 5,992 | 287 | 46.03 |
| ibm08 | 309 | 302 | 51,000 | 1,725 | 3,721 | 286 | 47.13 |
| ibm09 | 253 | 56 | 53,142 | 446 | 898 | 285 | 44.52 |
| ibm10 | 786 | 56 | 68,643 | 2,160 | 4,720 | 744 | 61.40 |
| ibm11 | 373 | 56 | 70,185 | 682 | 1,371 | 406 | 41.40 |
| ibm12 | 651 | 205 | 70,425 | 1,589 | 3,468 | 637 | 53.85 |
| ibm13 | 424 | 100 | 83,775 | 804 | 1,669 | 490 | 39.43 |
| ibm14 | 614 | 91 | 146,991 | 1,620 | 3,960 | 517 | 22.49 |
| ibm15 | 393 | 22 | 161,177 | 748 | 1,521 | 383 | 28.89 |
| ibm16 | 458 | 37 | 183,026 | 1,755 | 3,981 | 504 | 39.46 |
| ibm17 | 760 | 107 | 184,735 | 2,055 | 4,366 | 743 | 19.11 |
| ibm18 | 285 | 285 | 210,328 | 727 | 1,600 | 272 | 11.09 |

### A.7 Supplementary Experiment

**More benchmarks** We also conducted experiments in the IBM benchmark suite (ICCAD 2004) [31], which has been used to evaluate placement for more than a decade. We remove the "ibm05" because it does not contain any macros. We use our MaskPlace to place large macros and DREAMPlace

[6] github.com/limbo018/DREAMPlace
[7] github.com/google-research/circuit_training
[8] github.com/Thinklab-SJTU/EDA-AI

[9] to place standard cells. We compared our method with graph placement [3] and the simulated annealing method used in [3]. The results are in Table 15 and our method can achieve the lowest HPWL in all benchmarks.

Table 15: Comparisons of HPWL ($\times 10^5$) for macro and standard cell placement in ibm benchmark.

| Method | ibm01 | ibm02 | ibm03 | ibm04 | ibm05 | ibm06 |
|---|---|---|---|---|---|---|
| Graph Placement [3] | 31.71 | 55.12 | 80.00 | 86.86 | - | 63.48 |
| Simulated Annealing [3] | 25.85 | 54.87 | 80.68 | 83.32 | - | 69.09 |
| MaskPlace+DREAMPlace [9] | **24.18** | **47.45** | **71.37** | **78.76** | - | **55.70** |
| Method | ibm07 | ibm08 | ibm09 | ibm10 | ibm11 | ibm12 |
| Graph Placement [3] | 117.71 | 134.77 | 148.74 | 440.78 | 218.73 | 438.57 |
| Simulated Annealing [3] | 117.71 | 144.89 | 141.67 | 463.04 | 228.79 | 435.77 |
| MaskPlace+DREAMPlace [9] | **95.27** | **120.64** | **122.91** | **367.55** | **202.23** | **397.25** |
| Method | ibm13 | ibm14 | ibm15 | ibm16 | ibm17 | ibm18 |
| Graph Placement [3] | 278.93 | 455.31 | 520.06 | 642.08 | 814.37 | 450.67 |
| Simulated Annealing [3] | 259.89 | 405.80 | 510.06 | 614.54 | 720.40 | 442.00 |
| MaskPlace+DREAMPlace [9] | **246.49** | **302.67** | **457.86** | **584.67** | **643.75** | **398.83** |

For the larger circuit *bigblue4* in ISPD 2005 benchmark, the result of our method and baselines can been seen as Table 16. MaskPlace still achieved the best performance.

Table 16: HPWL ($\times 10^7$) results for *bigblue4* benchmark

| Benchmark | Random | NTUPlace3[6] | RePlAce[8] | DREAMPlace [9] |
|---|---|---|---|---|
| bigblue4 | 128.06±3.94 | 48.38 | 11.80±0.73 | 12.29±1.64 |
| Benchmark | Graph Placement [3] | DeepPR [22] | DeepPR-no-overlap [8] | MaskPlace |
| bigblue4 | 53.35±4.06 | 68.30±4.44 | 115.08±2.29 | **11.07±0.90** |

**Search time** We compared the search time of our method, Graph Placement [3] and DeepPR [22]. We tested all methods in the same environment and took the HPWL as the metric in benchmark *adaptec1*. The result is in Fig. 10. We can see that our approach can achieve the best performance in a few hours.

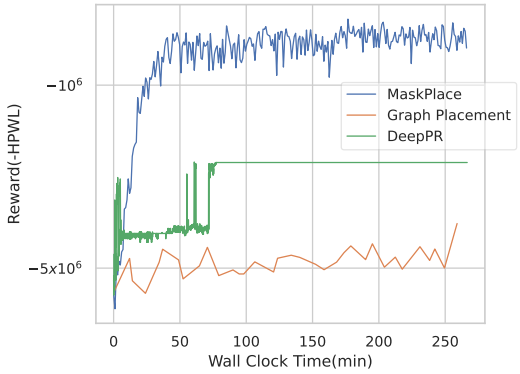



Figure 10: Search time comparison

### A.8 Detailed equation description of the model

We describe our model architecture in Fig. 4 in the form of equation.

With current state $s_t$, we first calculate the position masks $f_p^t, f_p^{t+1}$, wire masks $f_w^t, f_w^{t+1}$ and view mask $f_v^t$ via the mask generation function $m(\cdot)$.

$$f_p^t, f_p^{t+1}, f_w^t, f_w^{t+1}, f_v^t = m(s_t) \tag{3}$$

Then we extract the local feature $z_t^l$ via local mask fusion $g_\omega(\cdot)$ and the global features $z_t^g$ via the global mask encoder $enc_\eta(\cdot)$.

$$z_t^l = g_\omega(f_p^t, f_p^{t+1}, f_w^t, f_w^{t+1}) \tag{4}$$

where $g_\omega(\cdot)$ is a 1×1 convolutional neural network with parameter $\omega$.

$$z_t^g = enc_\eta(f_w^t, f_w^{t+1}, f_v^t) \tag{5}$$

where $enc_\eta(\cdot)$ is a convolutional neural network with ResNet-18 architecture with parameter $\eta$.

With local features $z_t^l$ and global features $z_t^g$, the state value $\hat{V}_t$ is derived by

$$\hat{V}_t = v_\phi(pos(t), z_t^g) \tag{6}$$

where $v_\phi$ is an MLP-like neural network with parameter $\phi$ and $pos(t)$ is an embedding vector which is related to step $t$.

We decode the global features $z_t^g$ into the dimension as same as the action space by the global mask decoder

$$z_t^{'g} = dec_\delta(z_t^g) \tag{7}$$

where $dec_\delta(\cdot)$ is a transpose convolutional neural network with parameter $\delta$.

Finally, we concatenate the local features $z_t^l$ and global features $z_t^{'g}$ in the channel dimension and merge them by another 1×1 convolutional neural network $\psi_\xi(\cdot)$. We further combine it with the position mask $f_p^t$ to generate action $a_t$ via the policy network $\pi_\theta(\cdot)$

$$a_t \sim \pi_\theta(\psi_\xi(z_t^l || z_t^{'g}), f_p^t) \tag{8}$$

where $\pi_\theta(\cdot)$ is an MLP-like neural network with parameter $\theta$ and $\psi_\xi(\cdot)$ is a 1×1 convolutional neural network with parameter $\xi$.

# B Related Work

**Classic optimization-based methods.** Optimization has been the dominant method in placement for decades. They can be divide into three categories: partitioning-based methods [4, 5], simulated annealing methods [10, 11] and analytical methods [6–9, 12–21].

Partitioning-based methods [4, 5] cluster the whole circuits into several parts to minimize the connections between parts. These methods first solve the placement problems within the same part and then place these parts to suitable positions on the chip based on the divide-and-conquer idea. However, optimizing modules within one part is an isolated problem, and sometimes it is hard to divide the circuits into relatively independent parts, which is highly related to the topology of the circuits.

Simulated Annealing (SA) methods [10, 11] are also known as hill-climbing methods, a widely used iterative heuristic algorithm for solving combinatorial optimization problems. They initialize a random status and then search for the following status by moving from the current status to a neighbor status. If the metrics of the neighbor status are better than that of the current status, they move to the neighbor status. Otherwise, the move may still be taken with a decreasing probability over time. The advantage is that they can be implemented when metrics do not have the analysis formula or cannot be differentiable. However, it is not efficient enough, and the placement results are highly dependent on the random initial state.

Analytical methods gradually replace the above two methods because of the best performance. They can be divided into quadratic methods [12–18] and nonlinear (non-quadratic) methods [6–9, 19–21]. Quadratic methods [12–18] transform the placement problem into a sequence of convex quadratic problems, and there are well-established solvers for such problems. However, it is a very rough approximation. Nonlinear methods [6–9, 19–21] design a single differentiable objective function and optimize it. The advantage is that it can handle large-scale modules. However, the objective function is still approximated, and they cannot avoid overlaps when combining multiple metrics in one objective function. Methods in this category achieved the highest placement quality among all classic methods [9].

**Learning-based methods.** With the development of deep learning, some learning-based approaches [11, 37–39] have been proposed to assist classic methods. Huang et al. [37] uses convolutional neural networks to estimate the congestion for SA placement. Vashisht et al. [11] uses the reinforcement learning models to generate the initial placement of SA. Kirby et al. [38], Agnesina et al. [39] help classic placement tools choose the most suitable hyperparameters with reinforcement learning methods. However, these methods do not implement end-to-end placement by deep learning, so the placement results depend heavily on the classic methods.

Pure reinforcement learning methods [3, 22, 23, 40] view placement as a process of placing modules sequentially. Mirhoseini et al. [3] uses reinforcement learning to place hard macros, and the force-directed method [18] to place remaining soft macros. Jiang et al. [23] replaces the force-directed method with DREAMPlace [9] to place soft macros based on Graph Placement [3]. Cheng and Yan [22] proposes a reinforcement learning method by using wirelength as the reward. Moreover, Chang et al. [40] puts all metrics in the RL reward. They have in common that they convert the circuit as a graph structure and input them to the graph neural networks [41]. However, the pin information has been lost, leading to sub-optimal placement. Also, they cannot avoid overlaps because of the reduction in search space. These methods still have room for improvement in terms of realistic chip placement. For instance, DeepPR [22] ignores the realistic size of the module. However, the size of the modules varies widely in most circuits. Although it proposes to use routing wirelength instead of HPWL as the reward, it will affect the efficiency and lead to sparse reward, making models hard to train. In contrast, HPWL is a high-quality wirelength estimation, and we do not need to discard this inherent dense reward.